06丨倾囊相授：我毕生所学的性能分析思路都在这里了

2019-12-27 高楼

讲述：高楼
时长 43:26 大小 24.86M

我还年轻的时候，经常听一些大会或者演讲。有些人说，思路逻辑非常重要。我那时就想，你肯定是瞎忽悠的，因为我怎么就没听懂你说的思路呢？

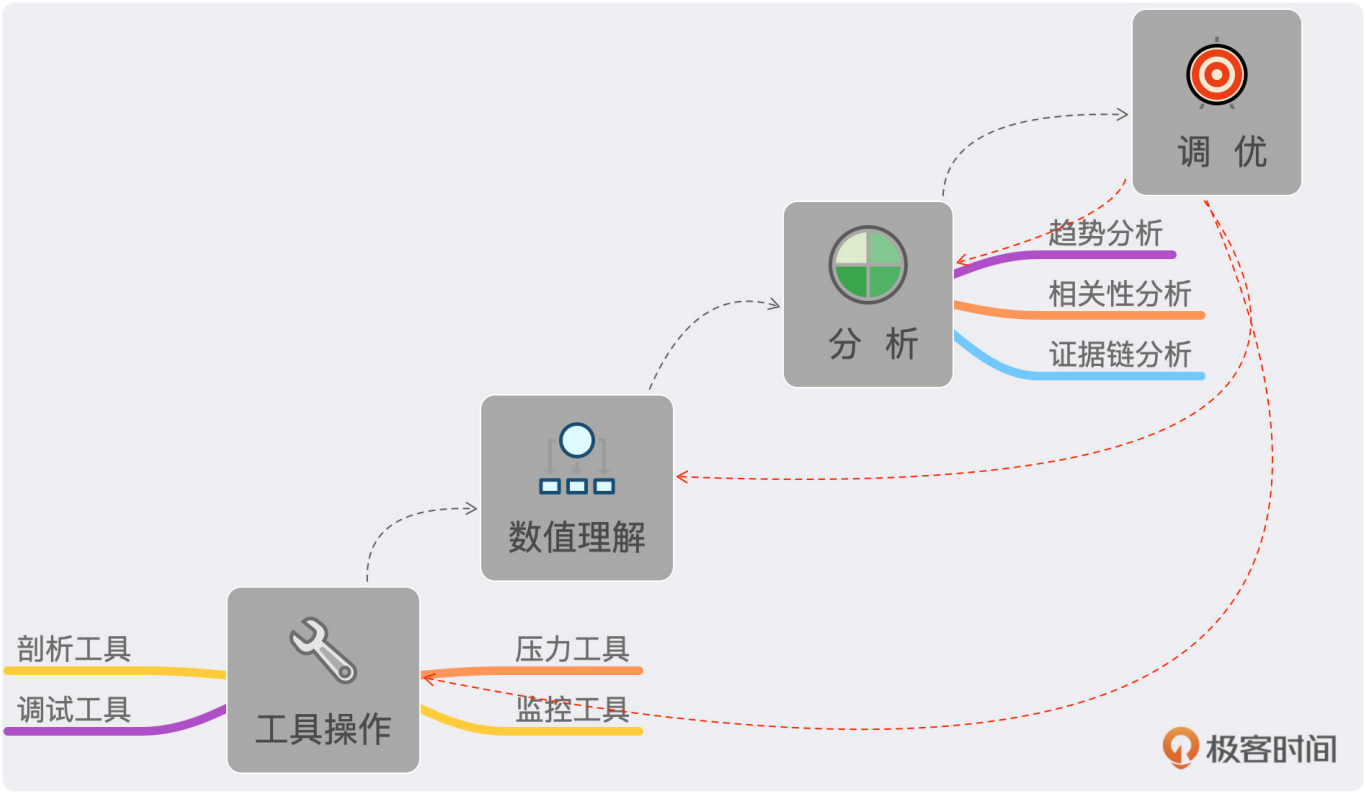
而现在轮到自己来写或者讲一些东西的时候，才发现他们说得很对，而我之所以不理解，也是有原因的。

性能分析思路和具体的实现之间，有一道鸿沟，那就是操作的能力。之前我为什么听不懂那些人的思路，其实是因为我没有操作的功底。

而有了操作的功底之后，还有一个大的鸿沟要越过去，那就是从操作到对监控计数器的理解。这一步可以说让很多性能测试人员都望而却步了。

但是这还不算完，这一步迈过去之后，还有一个跳跃，就是相关性分析和证据链分析的过程。

如此一来，就会得到一张**性能测试分析的能力阶梯视图**，如下：



1. 工具操作：包括压力工具、监控工具、剖析工具、调试工具。
2. 数值理解：包括上面工具中所有输出的数据。
3. **趋势分析、相关性分析、证据链分析**：就是理解了工具产生的数值之后，还要把它们的逻辑关系想明白。这才是性能测试分析中最重要的一环。
4. 最后才是调优：有了第 3 步之后，调优的方案策略就有很多种了，具体选择取决于调优成本和产生的效果。

那么怎么把这些内容都融会贯通呢？下面我们就来说说性能测试分析的几个重要环节。

应该说，从我十几年的性能工作中，上面讲的这些内容是我觉得最有价值的内容了。在今天的文章中，我们将对它做一次系统的说明。我先把**性能分析思路**大纲列在这里：

1. 瓶颈的精准判断；
2. 线程递增的策略；

3. 性能衰减的过程；

4. 响应时间的拆分；

5. 构建分析决策树；

6. 场景的比对。

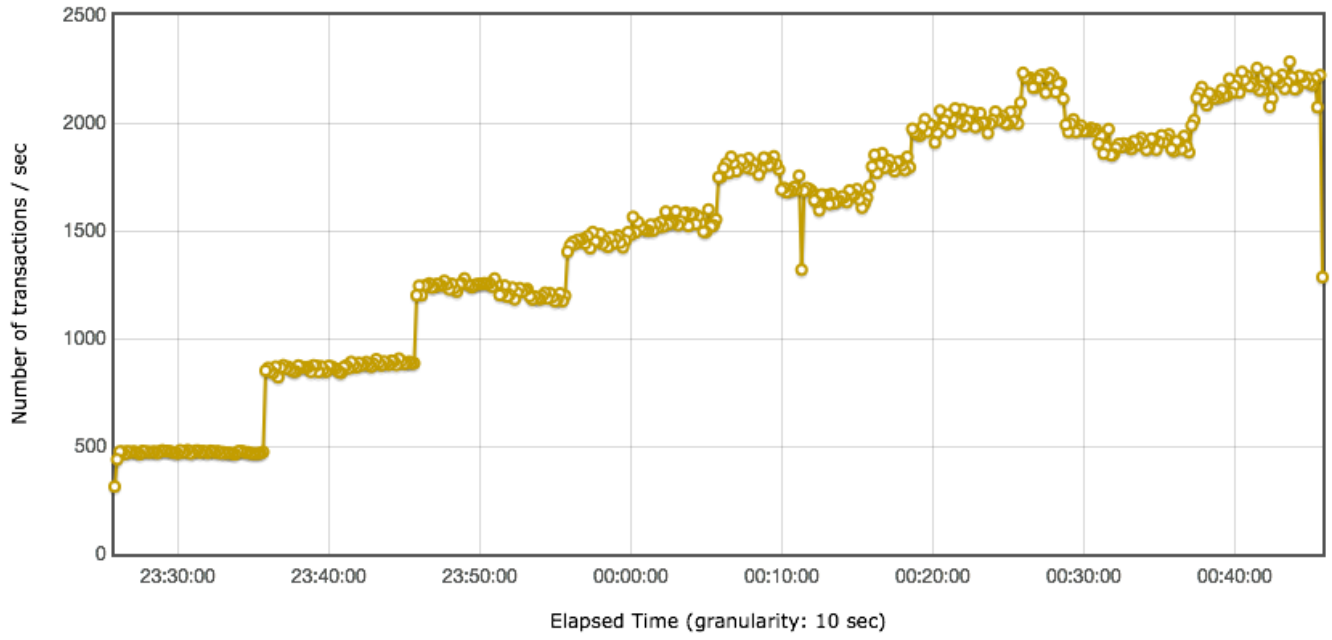
瓶颈的精准判断

TPS 曲线

对性能瓶颈做出判断是性能分析的第一步，有了问题才能分析调优。

之前有很多人在描述性能测试的过程中，说要找到性能测试中曲线上的“拐点”。我也有明确说过，大部分系统其实是没有明确的拐点的。

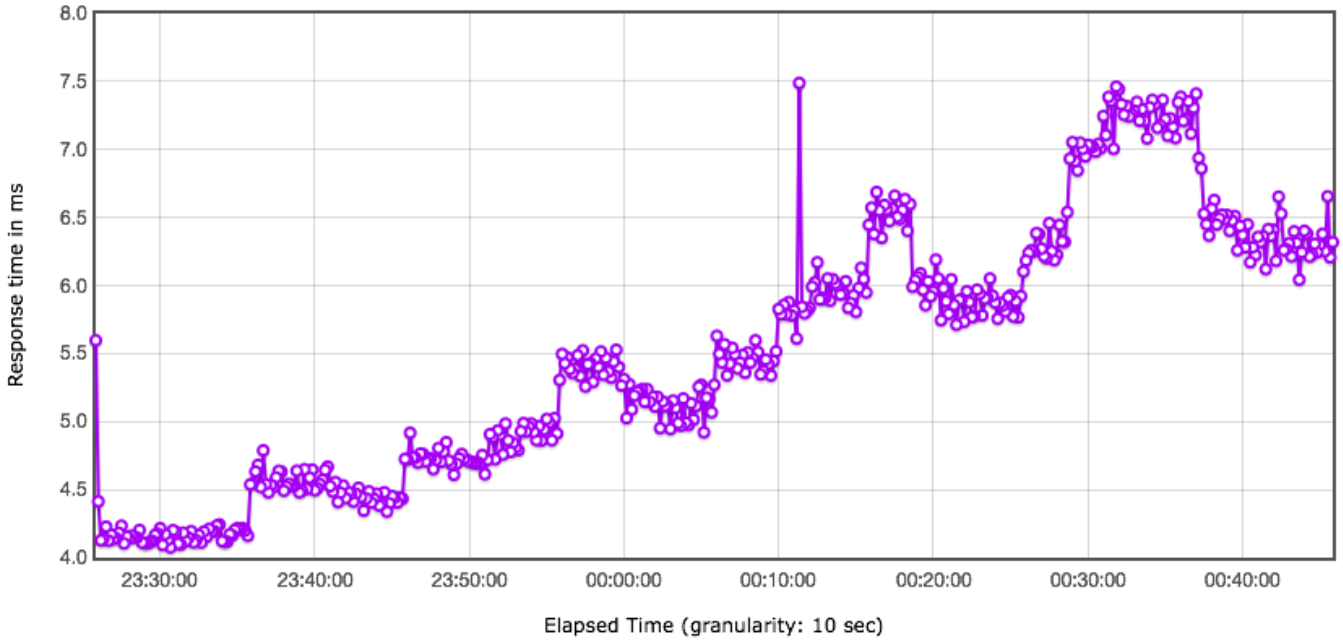
举例来说，TPS 的视图如下：



TPS 图 1

显然，这是一个阶梯式增加的场景，非常好。但是拐点在哪呢？有人说，显然在 1200TPS 左右的时候。也有人说了，显然是到 1500TPS 才是拐点呀。但是也有人说，这都已经能到 2000TPS 了，显然 2000TPS 是拐点。

我们再来看一下这张图对应的响应时间视图：



响应时间图 1

是不是有人要说响应时间为 4.5ms 时是拐点了？

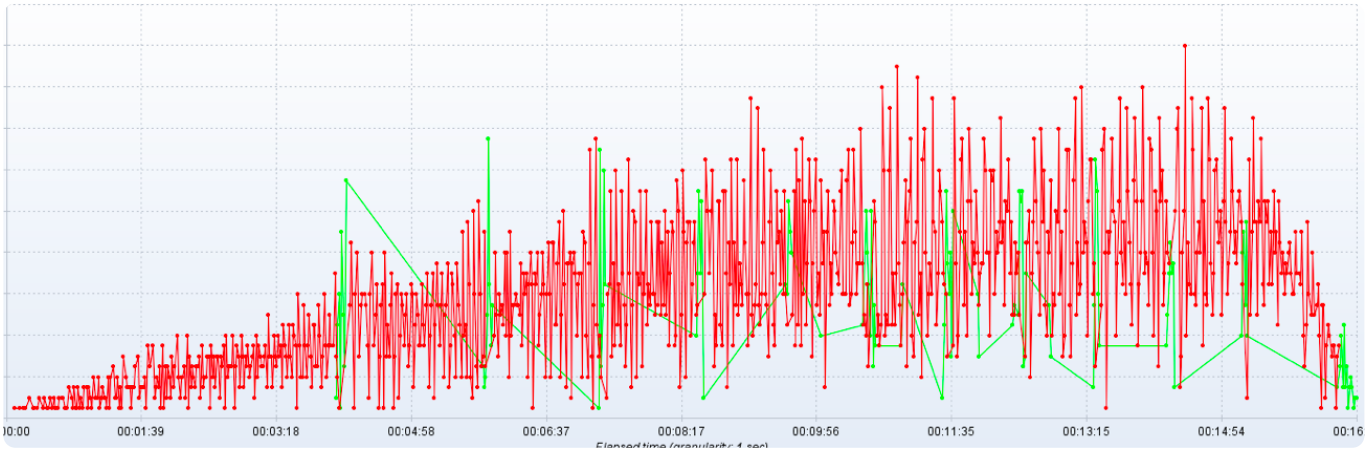
其实这些对拐点的判断，都是不合理的。如果我们对 TPS 的增加控制得更为精准的话，那么这个 TPS 的增加是有一个有清晰的弧度，而不是有一个非常清晰的拐点。

但是至少我们可以有一个非常明确的判断，那就是瓶颈在第二个压力阶梯上已经出现了。因为响应时间增加了，TPS 增加得却没有那么多，到第三个阶梯时，显然增加的 TPS 更少了，响应时间也在不断地增加，所以，性能瓶颈在加剧，越往后就越明显。

那么我们的判断就是：

1. 有瓶颈！
2. 瓶颈和压力有关。
3. 压力呈阶梯，并且增长幅度在衰减。

如果你觉得上面的瓶颈还算清晰的话，那么我们再来看一张图：



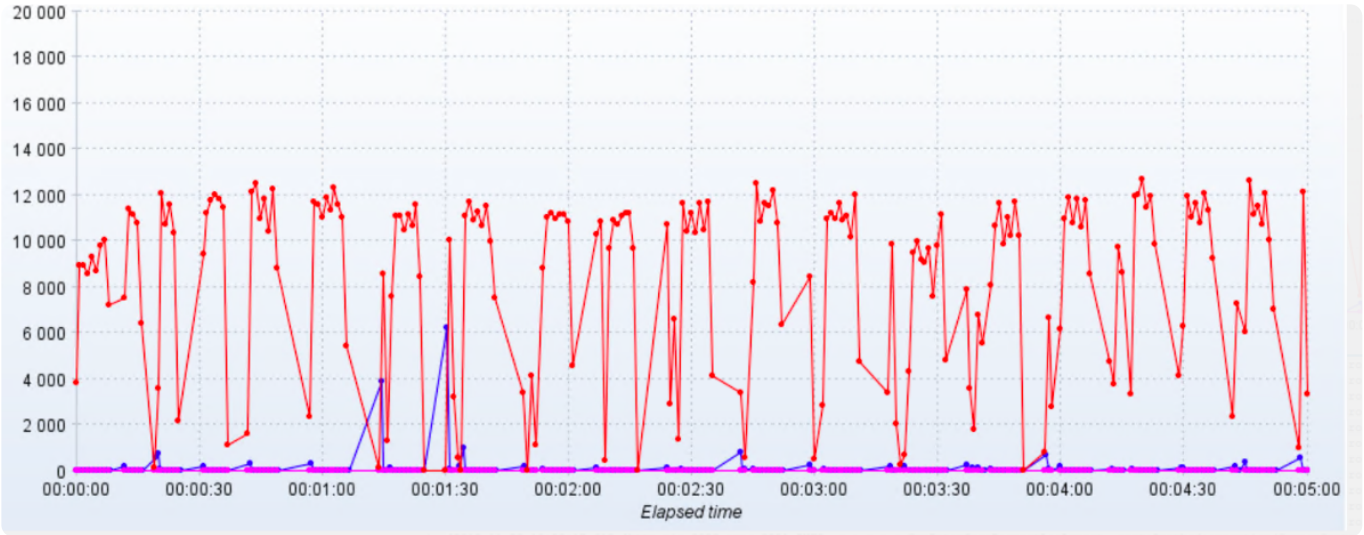
TPS 图 2

在这个 TPS 的曲线中，你还能判断出拐点在哪吗？

显然是判断不出来拐点的，但是我们根据图得出以下几个结论：

1. 有瓶颈！
2. 瓶颈和压力有关。
3. 压力也是阶梯的，但是并没有明确的拐点。

我们再来看一个 TPS 图：



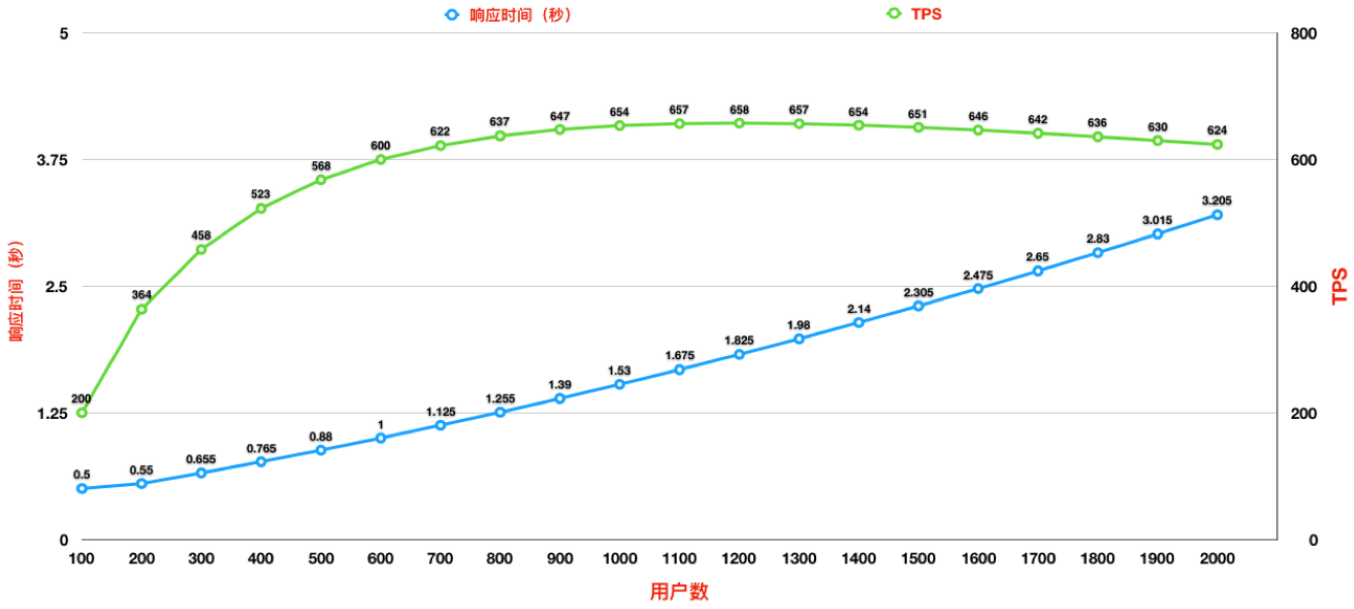
TPS 图 3

看到这张图，是不是明显感觉系统有瓶颈呢？那么瓶颈是不是和压力大小有关呢？

这种比较有规律的问题，显然不是压力大小的原因。为什么呢？因为 TPS 周期性地出现降低，并且最大的 TPS 也都恢复到了差不多的水位上。所以，即使是压力降低，也最多降低

最大的 TPS 水位，会让问题出现得更晚一点，但是不会不出现。

综合以上，如果画一个示意图的话，TPS 的衰减过程大概会如下所示：



1. 随着用户数的增加，响应时间也在缓慢增加。
2. TPS 前期一直都有增加，但是增加的幅度在变缓，直到变平。

在这样的趋势图中，我们是看不到明确的拐点的。但是我们能做的清晰的判断就是：有瓶颈！

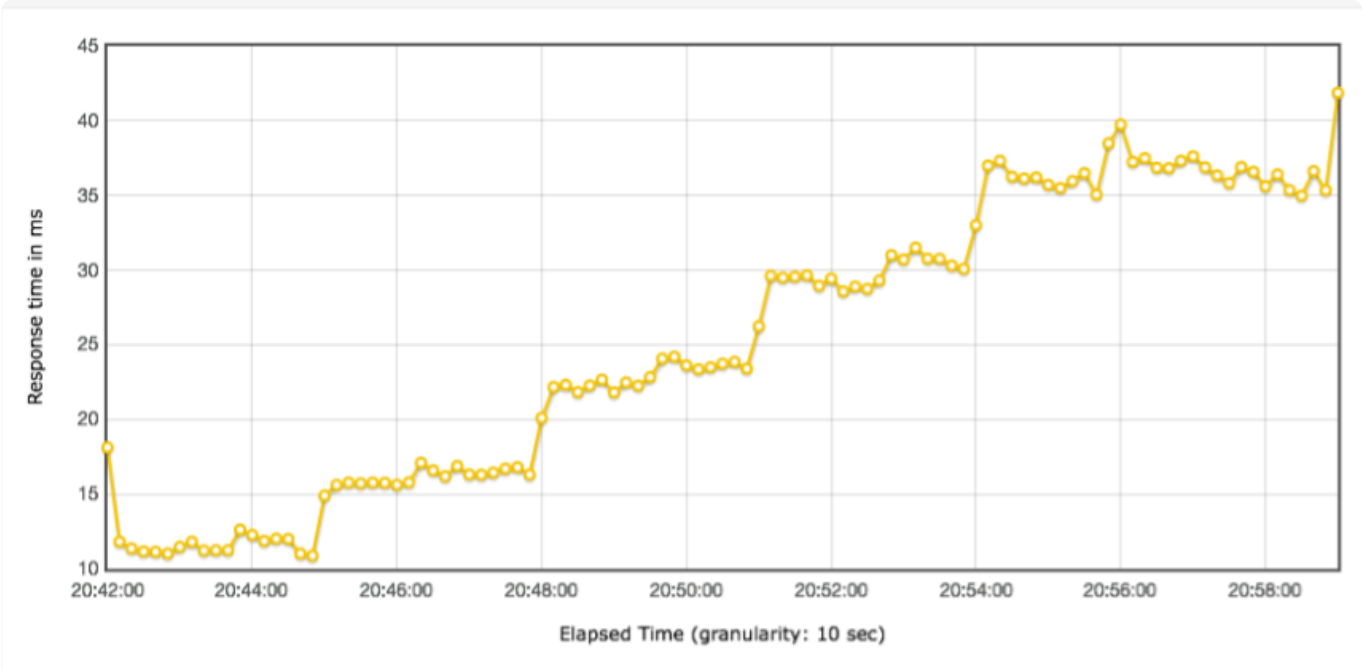
所以对 TPS 曲线来说，它可以明确告诉我们的就是：

1. 有没有瓶颈：其实准确说所有的系统都有性能瓶颈，只看我们在哪个量级在做性能测试了。
2. 瓶颈和压力有没有关系：TPS 随着压力的变化而变化，那就是有关系。不管压力增不增加，TPS 都会出现曲线趋势问题，那就是无关。

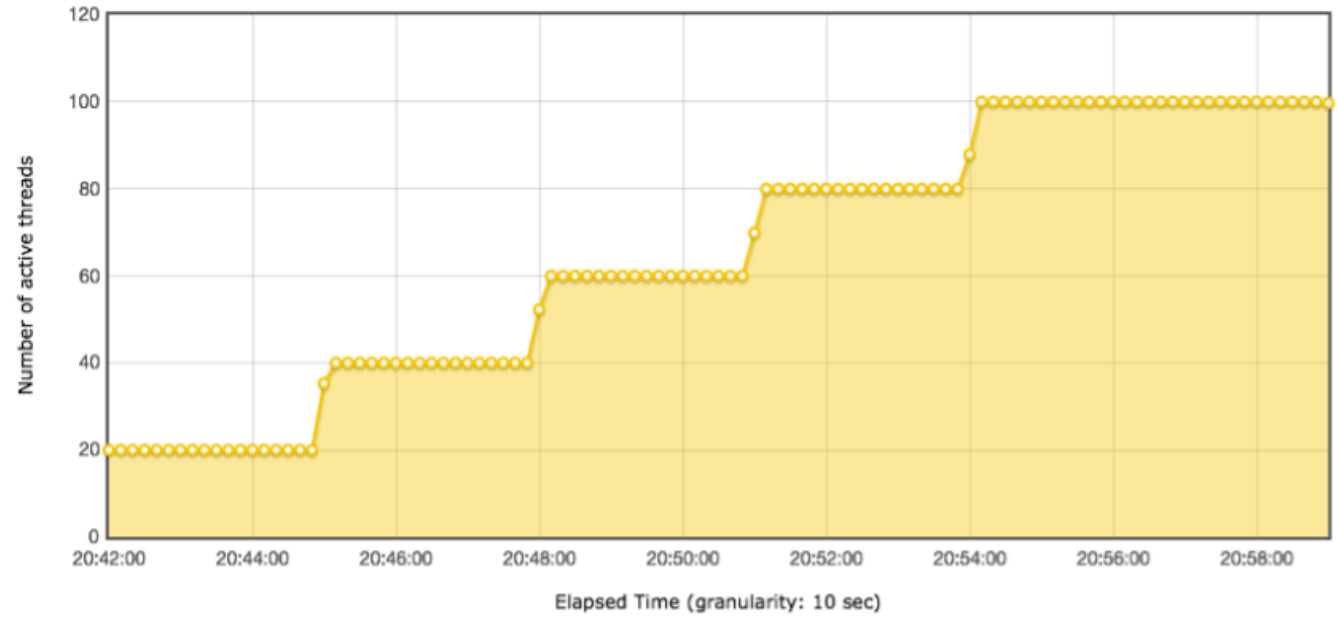
这时你可能会问，为什么不看响应时间就武断地下此结论呢？其实响应时间是用来判断业务有多快的，而 TPS 才是用来判断容量有多大的。

响应时间的曲线

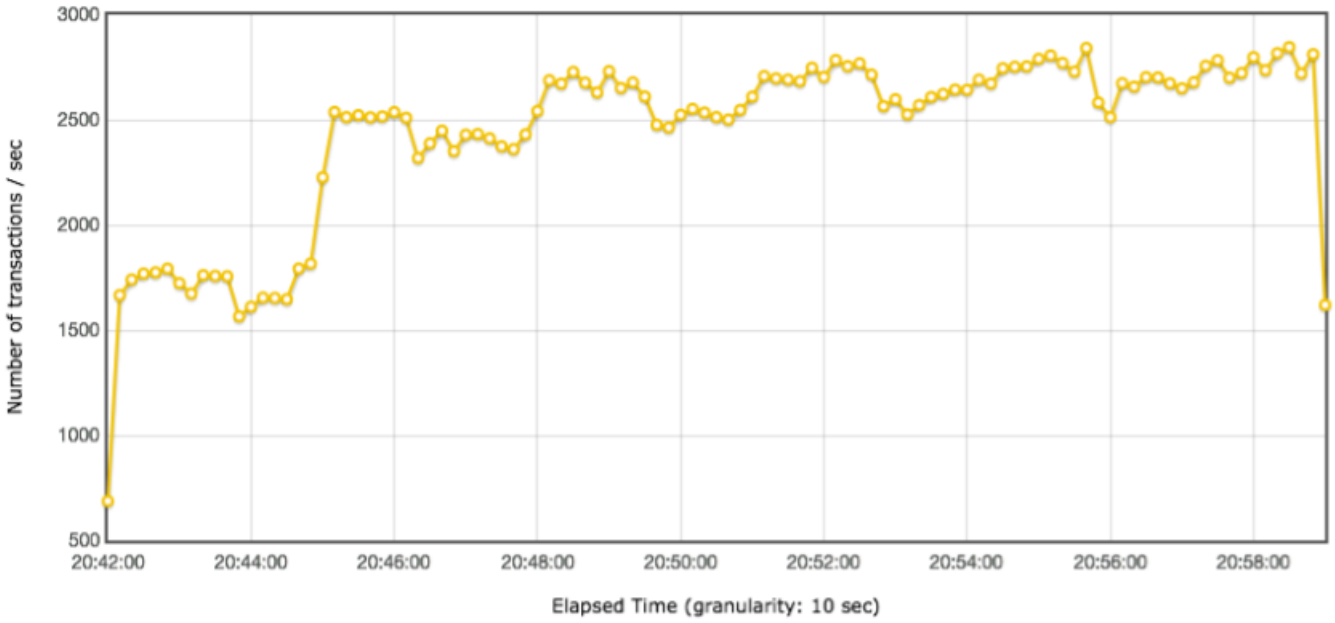
我们还是来看看响应时间，下面看一张响应时间图：



它对应的线程图是：



多明显的问题，随着线程的增多，响应时间也在增加，是吧。再来看它们对应的 TPS 图：



到第 40 个线程时，TPS 基本上达到上限，为 2500 左右。响应时间随着线程数的增加而增加了，系统的瓶颈显而易见地出现了。

但是，如果只让你看 TPS 曲线，你是不是也会有同样的判断？那就是：有瓶颈！并且和压力有关？所以说，其实 TPS 就可以告诉我们系统有没有瓶颈了，而响应时间是用来判断业务有多快的。

后面我们还会提到响应时间会是性能分析调优的重要分析对象。

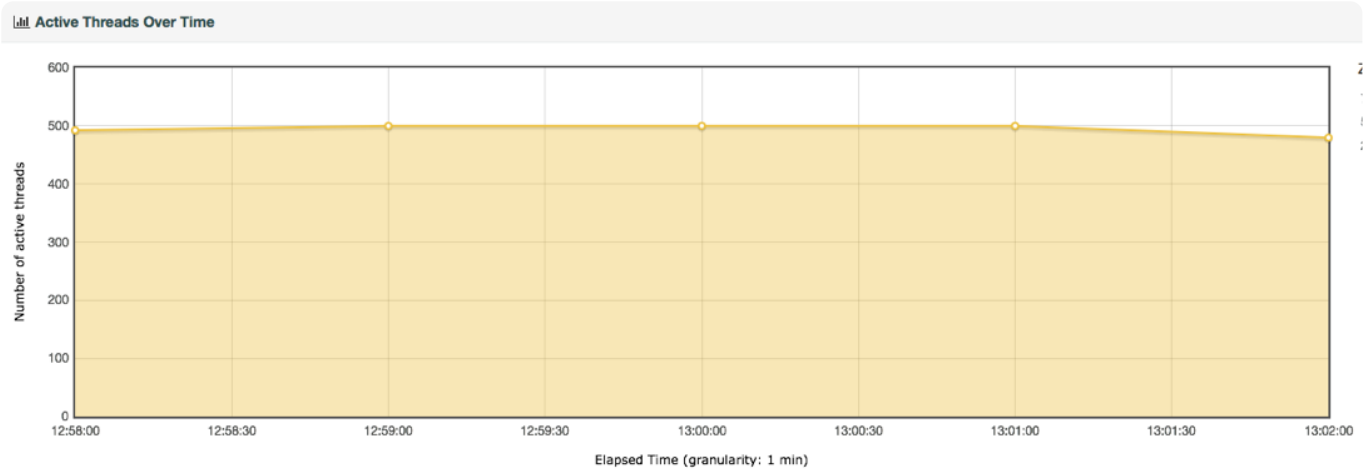
线程递增的策略

讲完响应时间之后，我们再来看下线程递增。

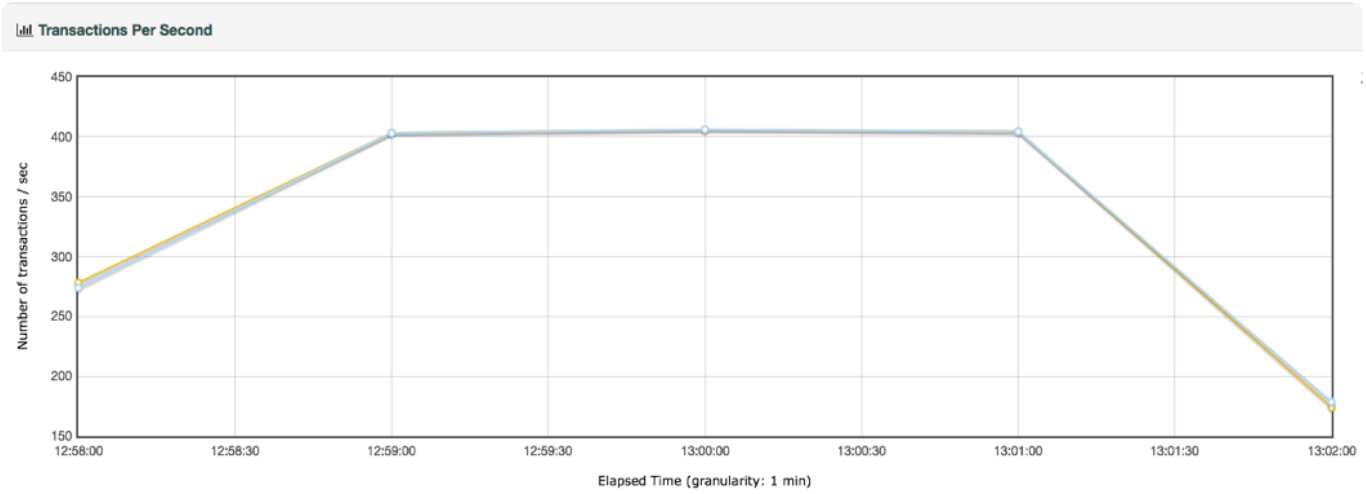
在见识了很多性能测试人员做的场景之后，必须得承认，有些场景的问题太多了。

首先，我们来看两个场景的执行对比。

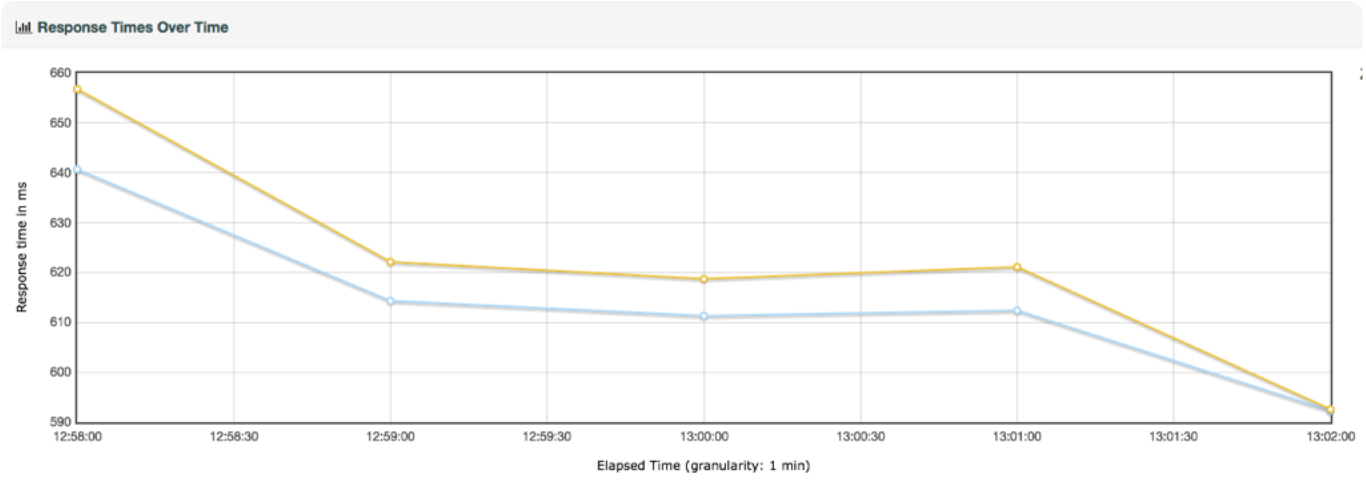
场景 1 的线程图：



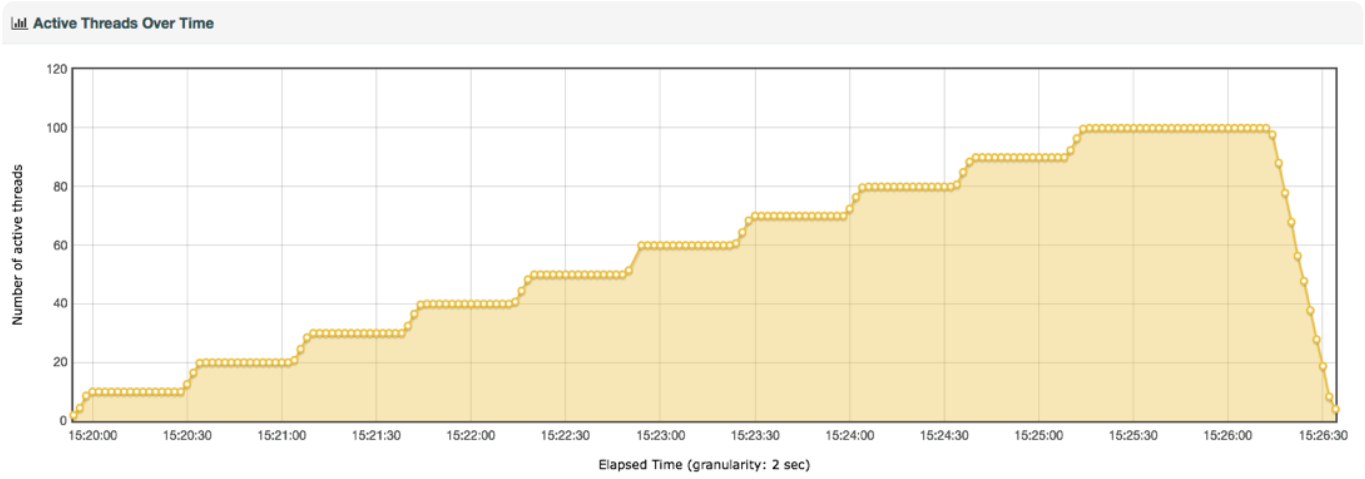
场景 1 的 TPS 图：



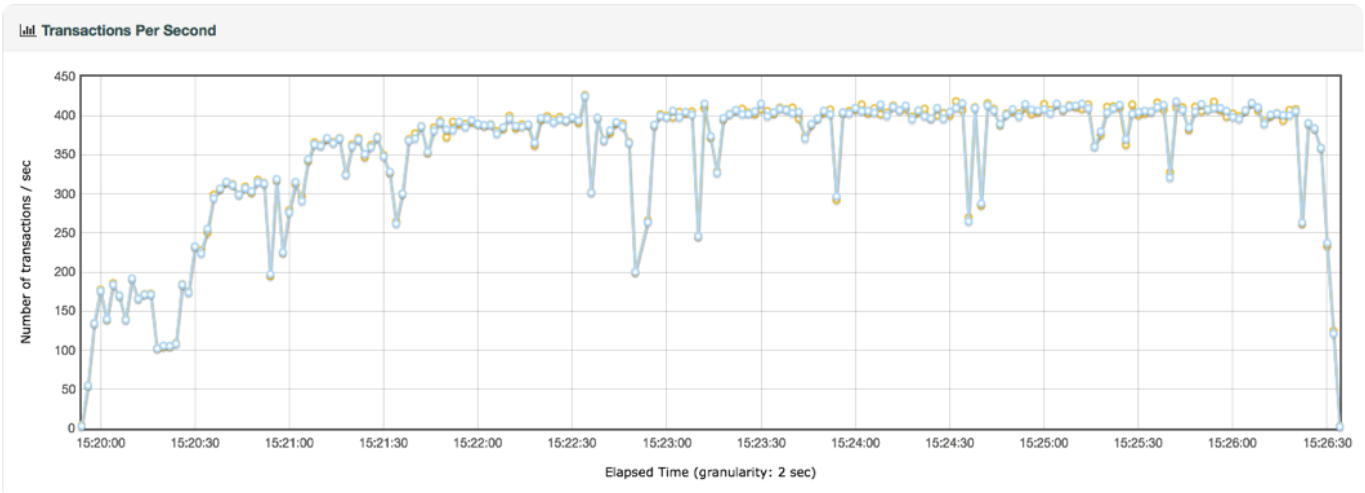
场景 1 的响应时间图：



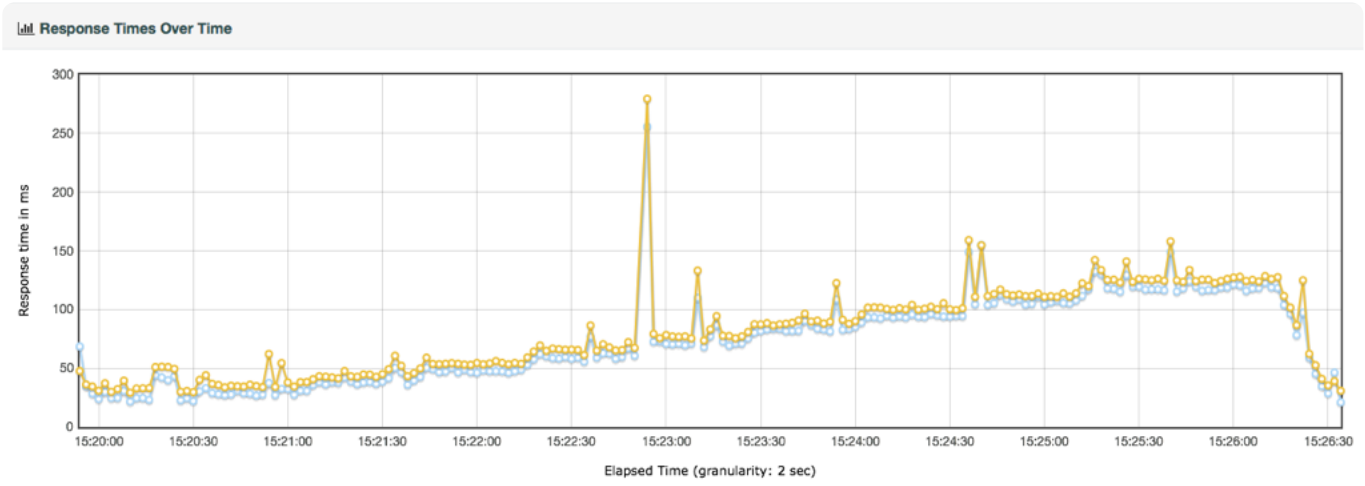
场景 2 的线程图：



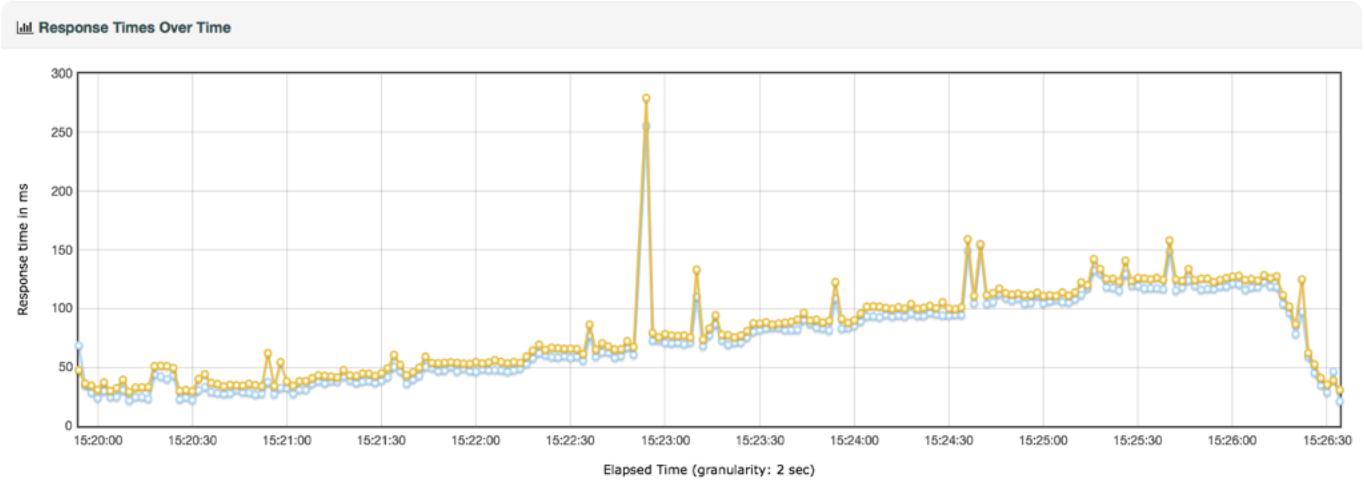
场景 2 的 TPS 图：



场景 2 的响应时间图：



这两个场景的比对如下：



对比项	场景1	场景2
线程	一次性加到500	10线程递增，并且递增中也是有梯度的
TPS	最大值达到400	最大值达到400，但有递增过程中有抖动
响应时间	在600ms-660ms之间	在20ms-150ms之间
错误率	无	无
粒度	以分钟为粒度	以2秒为粒度

有了这些对比数据之后，你是不是觉得哪里似乎是有问题的?

对的!

TPS 都是达到 400，但两个场景中线程递增的策略不同，产生的响应时间完全不同。虽然都没有报错，但是第一种场景是完全不符合真实的业务场景的。这是为什么呢?

在场景的执行过程中，首先，响应时间应该是从低到高的，而在场景 1 中不是这样。其次，线程应该是递增的，而场景 1 并没有这样做（这里或许有人会想到秒杀的场景，认为场景 1 符合秒杀的业务设定，这个问题我们稍后提及）。最后，在两个场景中，TPS 的上限都达到了 400TPS。但是你可以看到，在场景 2 中，只要 40 个线程即可达到，但场景 1 中居然用到了 500 线程，显然压力过大，所以响应时间才那么长。

其实在生产环境中，像场景 1 这样的情形是不会出现的。如果它出现了，那就是你作为性能测试的责任，因为你没有给出生产环境中应该如何控制流量的参数配置说明。

同时，我们从上面的场景对比可以看到，**对一个系统来说，如果仅在改变压力策略（其他的条件比如环境、数据、软硬件配置等都不变）的情况下，系统的最大 TPS 上限是固定的**。

场景 2 使用了递增的策略，在每个阶梯递增的过程中，出现了抖动，这就明显是系统设置的不合理导致的。设置不合理，有两种可能性：1. 资源的动态分配不合理，像后端线程池、内存、缓存等等；2. 数据没有预热。

我们再回到之前说的秒杀场景。

说到秒杀场景，有人觉得用大线程并发是合理的，其实这属于认识上的错误。因为即使线程数增加得再多，对已经达到 TPS 上限的系统来说，除了会增加响应时间之外，并无其他作用。所以我们描述系统的容量是用系统当前能处理的业务量（你用 TPS 也好，RPS 也好，HPS 也好，它们都是用来描述服务端的处理能力的），而不是压力工具中的线程数。这一点，我在第 5 篇文章中已经做了详细的解析，你可以回去再看看。

那么，对于场景中线程（有些工具中叫虚拟用户）递增的策略，我们要做到以下几点：

1. 场景中的线程递增一定是连续的，并且在递增的过程中也是有梯度的。
2. 场景中的线程递增一定要和 TPS 的递增有比例关系，而不是突然达到最上限。后面在场景的篇幅中我们会再说它们之间的比例关系。
3. 上面两点针对的是常规的性能场景。对于秒杀类的场景，我们前期一定是做好了系统预热的工作的，在预热之后，线程突增产生的压力，也是在可处理范围的。这时，我们可以设计线程突增的场景来看系统瞬间的处理能力。如果不能模拟出秒杀的陡增，就是不合理的场景。

这里给出我做性能场景递增的经验值：

响应时间	递增幅度
0-50ms	1
50-100ms	1-3
100-200ms	3-5
200-500ms	5-10

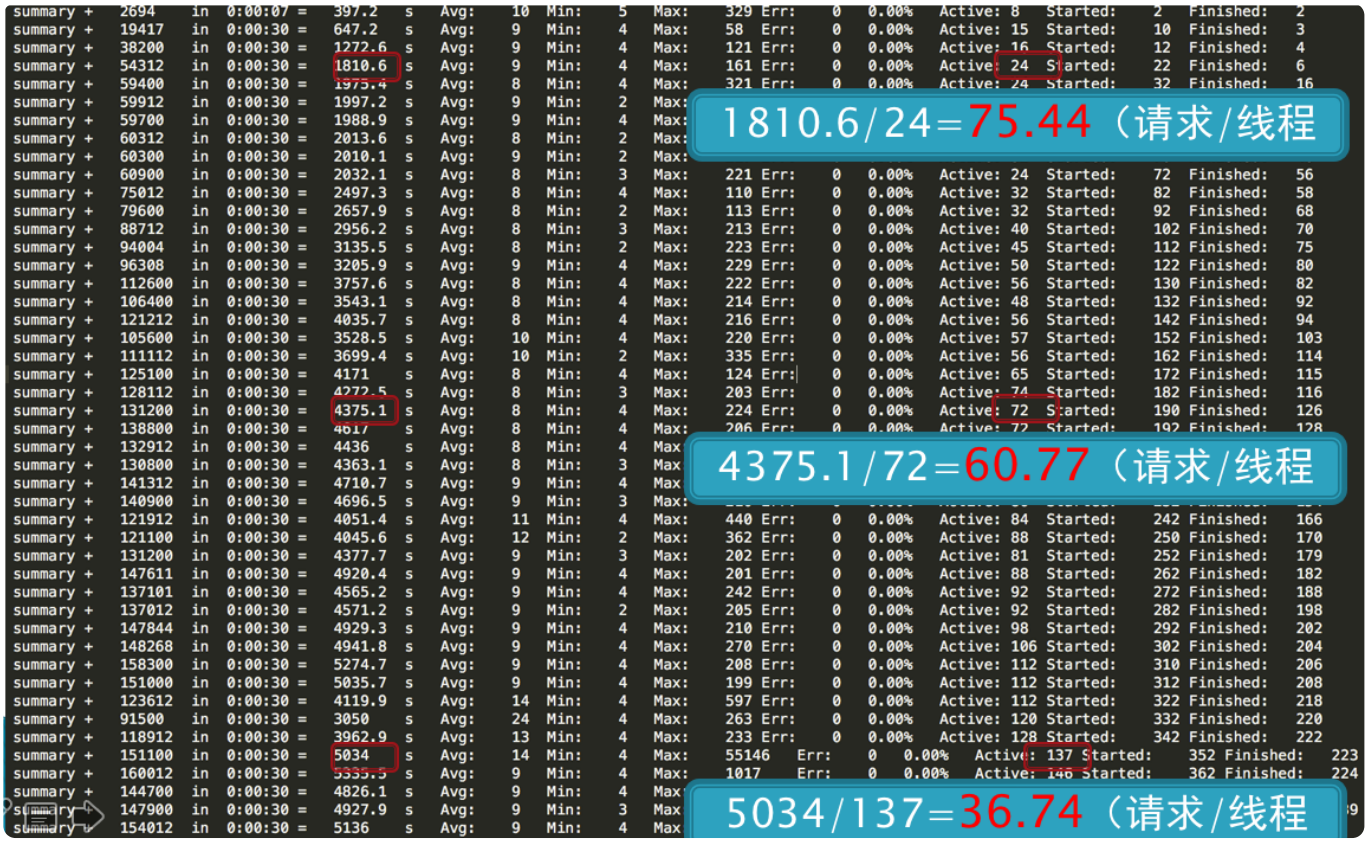
当然这里也不会是放在哪个系统中都适合的递增幅度，你还是要根据实际的测试过程来做相应的判断。

有了这些判断之后，相信大家都能做出合理的场景来了。

性能衰减的过程

有了瓶颈的判断能力，也有了线程递增的意识，那么下面在场景执行中，我们就要有判断性能衰减的能力了吧。

来，我们先看一个压力过程中产生的结果图。



在递增的压力过程中，随着用户数的增加。我们可以做几次计算。

第一次计算，在线程达到 24 时，TPS 为 1810.6，也就是每线程每秒发出 75.44 个请求。

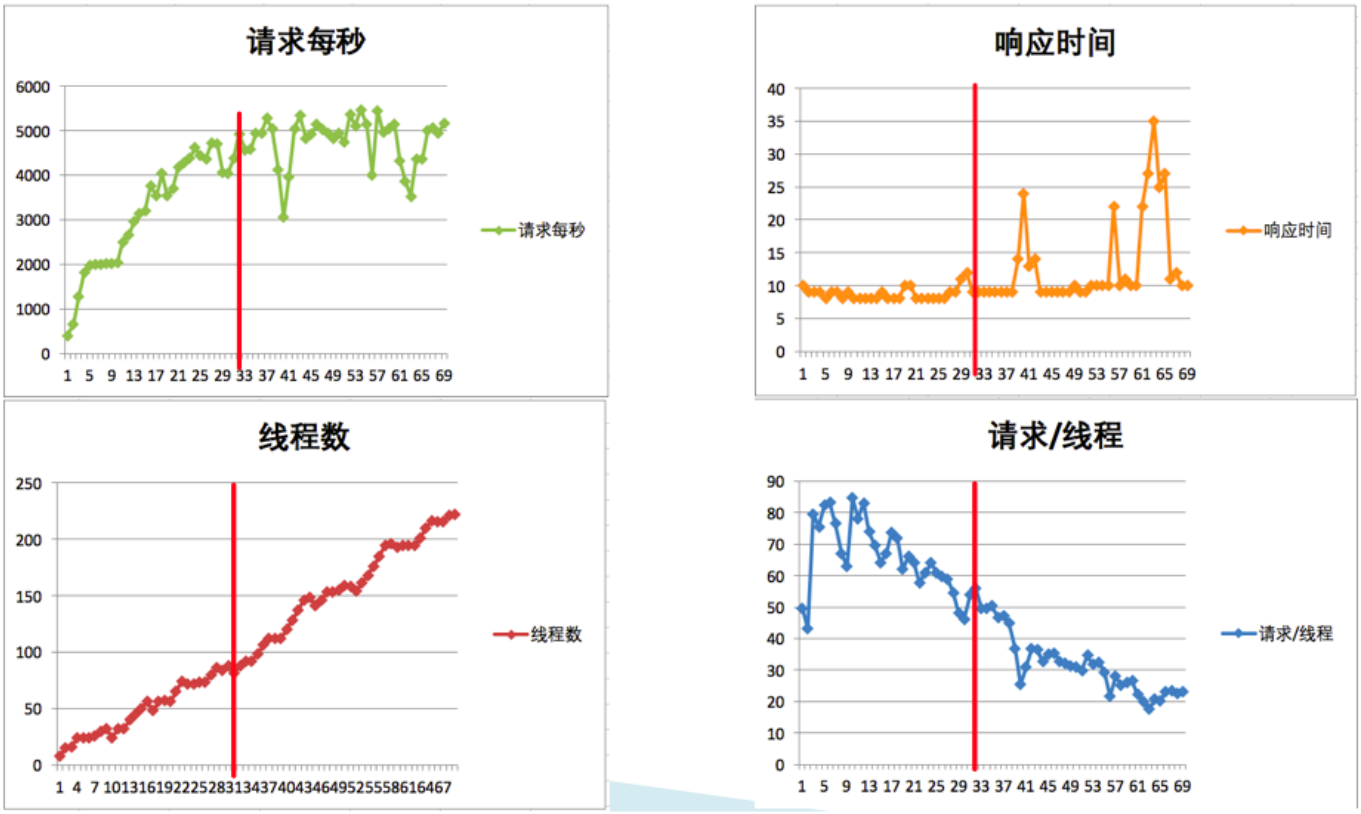
第二次计算，在线程达到 72 时，TPS 为 4375.1，也就是每线程每秒发出 60.77 个请求。

第三次计算，在线程达到 137 时，TPS 为 5034，也就是每线程每秒发出 36.74 个请求。

通过这三次计算，我们是不是可以看到，每线程每秒发出的请求数在变少，但是整体 TPS 是在增加的。

我们有很多做性能测试的人，基本上，只看 TPS 和响应时间的时候，在上面这个示例中，肯定会一直往上加用户。虽然响应时间在增加，但是增加得也不多嘛。

但实际上，通过我们的计算可以知道，性能是在不断地衰减的。我们来看一张统计图：



通过红线的大致比对可以知道，当每线程每秒的请求数降到 55 左右的时候，TPS 就达到上限了，大概在 5000 左右，再接着往上增加线程已经没有用了，响应时间开始往上增加了。

这就是性能衰减的过程（题外话，在上图中，其实还有一个问题，就是在红线前面，性能在上升的过程中有几次抖动，这个抖动到后面变大了，也变频繁了，如果这是必然出现的抖动，那也是配置问题，希望你注意到这一点）。

为什么要这么细致地描述性能衰减的过程呢？

其实我就是想告诉你，**只要每线程每秒的 TPS 开始变少，就意味着性能瓶颈已经出现了。但是瓶颈出现之后，并不是说服务器的处理能力（这里我们用 TPS 来描述）会下降，应该说 TPS 仍然会上升，在性能不断衰减的过程中，TPS 就会达到上限。**

这也是前面我说的，性能瓶颈其实在最大 TPS 之前早就已经出现了。

那么我们是不是应该在性能衰减到最大 TPS 时就停止场景呢？这个不一定的哦。

因为停不停场景，取决于我们的场景目标，如果我们只是为了得到最大 TPS，那确实可以停止场景了。但是，如果我们要扩大化性能瓶颈，也就是说为了让瓶颈更为明显，就完全不需要停止场景，只要不报错，就接着往上压，一直压到我们要说的下一个话题——响应时间变长，需要拆分。

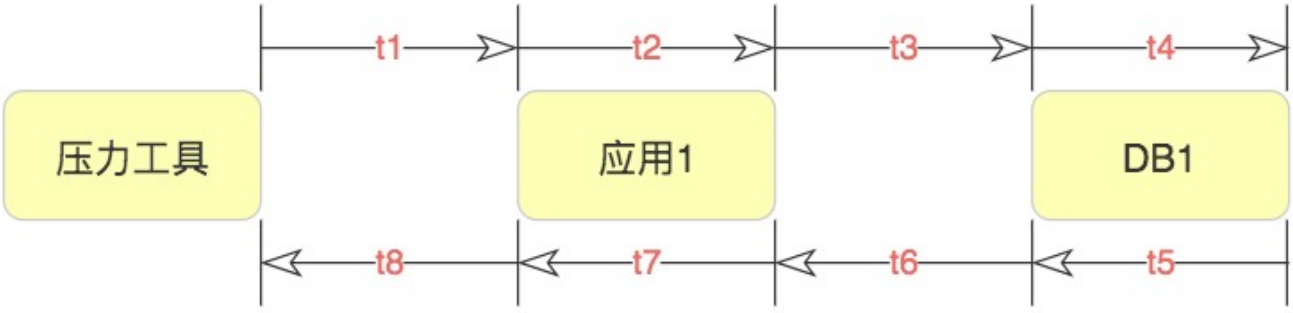
响应时间的拆分

在性能分析中，响应时间的拆分通常是一个分析起点。因为在性能场景中，不管是什么原因，只要系统达到了瓶颈，再接着增加压力，肯定会导致响应时间的上升，直到超时为止。

在判断了瓶颈之后，我们需要找到问题出现在什么地方。在压力工具上看到的响应时间，都是经过了后端的每一个系统的。

那么，当响应时间变长，我们就要知道，它在哪个阶段时间变长了。

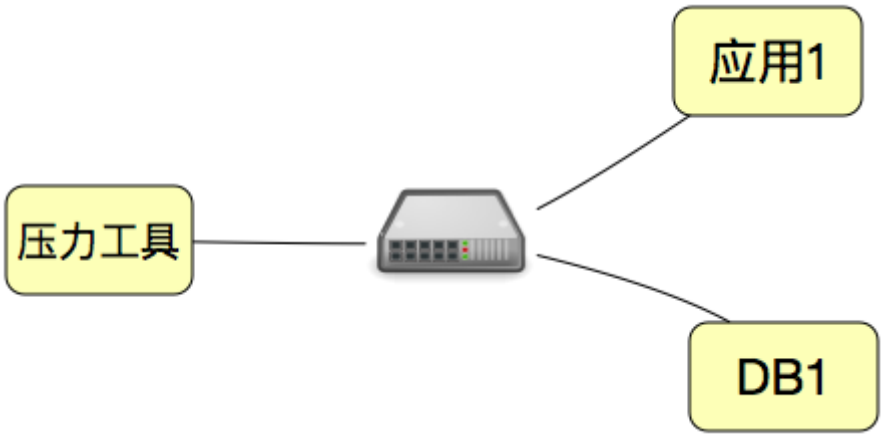
我们看下这张图。



这应该是最简单的一个压力测试逻辑了。一个应用，一个 DB，结果也拆分出了 8 个时间段，这还是在我没有加上压力工具自己所消耗的时间的情况下。

如果我们要分析压力工具中的响应时间，拆分的逻辑就是上面这个示意图。

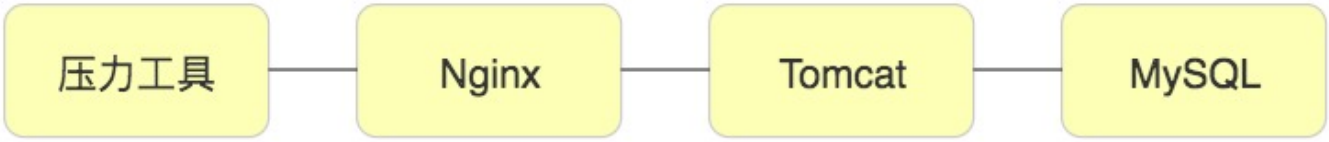
但是在真实的场景中，基本上不是这样的。如果是内网，那基本上都是连在一个交换机上，所以通常是这样的：



在这样的拓扑中，我们仍然可以拆出来 t1 到 t8 的时间。只是实际动手的时候，思路一定要清晰，时间拆分是从哪里到哪里，要画出来，不能混乱。

我们有很多手段可以进行时间的拆分，当然要看我们的应用支持哪一种。

如果我们是这样的架构，拆分时间应该是比较清楚的。



首先我们需要查看 Nginx 上的时间。日志里就可以通过配置 $request_time$ upstream_response_time 得到日志如下信息：

```
1 14.131.17.129 - - [09/Dec/2019:08:08:09 +0000] "GET / HTTP/1.1" 200 25317 0.02
```

最后两列中，前面是请求时间的 28ms，后面是后端响应时间的 28ms。

同时，我们再到 Tomcat 上去看时间。

```
1 172.18.0.1 - - [09/Dec/2019:08:08:09 +0000] "GET / HTTP/1.1" 200 25317 28 27 h
```

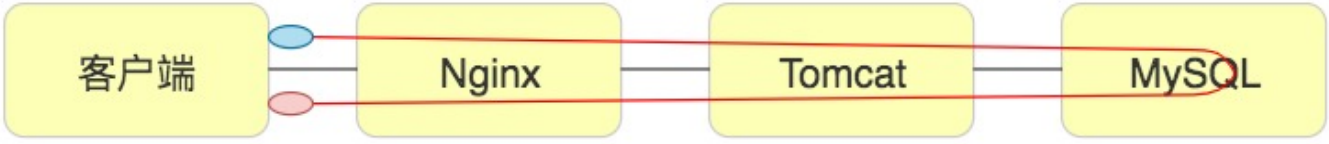
复制代码

请求时间消耗了 28ms，响应时间消耗了 27ms。

接着再来看一下前端的时间消耗。

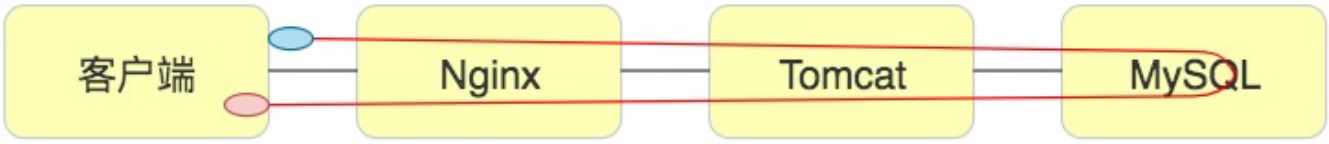
从这里可以看到，从发出请求到接收到第一个字节，即 TTFB 是 55.01ms，内容下载用了 11.75ms。从这就可以看得出 Nginx 基本上没消耗时间，因为它和 Tomcat 上的请求响应时间非常接近。

那么网络上的消耗时间怎么样呢？我看到有很多人用 TTFB 来描述网络的时间。先来说明一下，TTFB 中显然包括了后端一系列处理和网络传输的时间。如下图所示。



下面的红色点是指要接收的内容。上面的红色线就是 TTFB。

如果接收完了呢？就是这个状态。

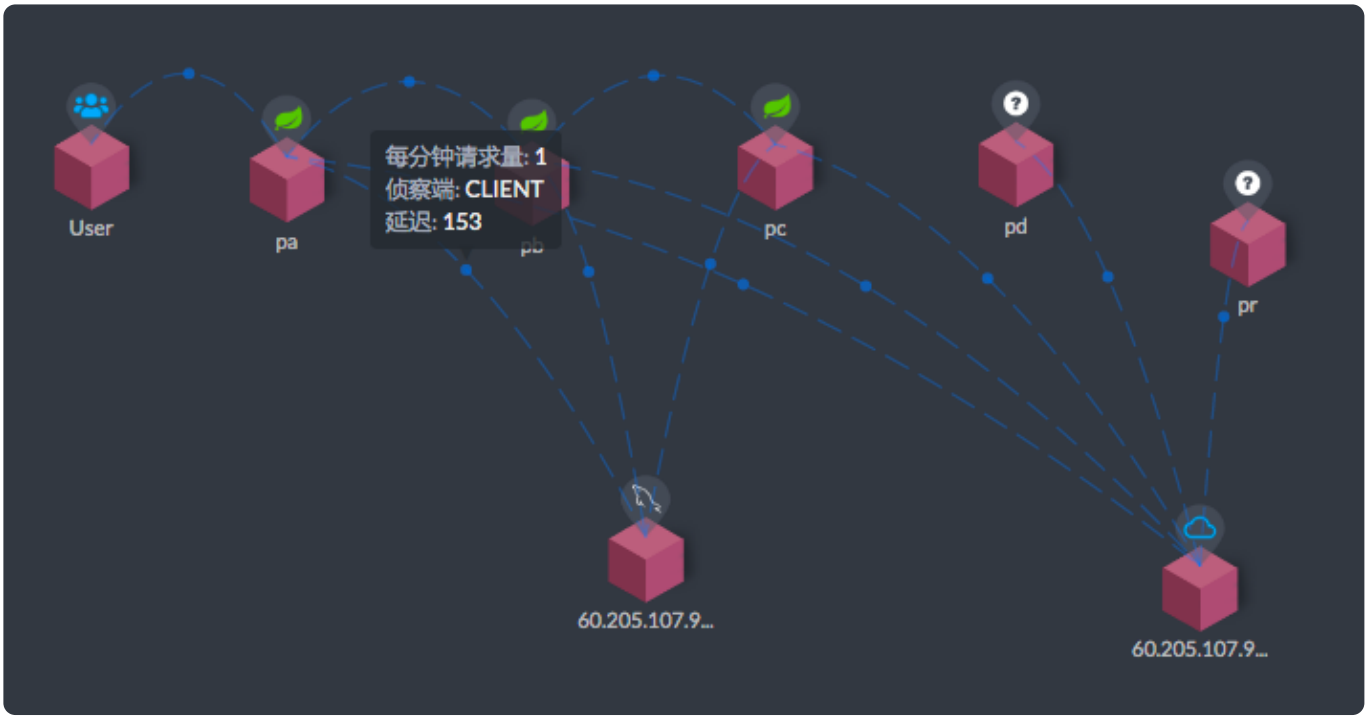


所以，我觉得用 TTFB 描述网络的健康状态并不合理。如果用 Content Download 来描述会更为合理。比如我们上面的这个例子中，那就是 11.75ms 下载了 25317 Bytes 的内容。

Tomcat 上基本上是消耗了处理的所有时间，当然这中间也包括了 MySQL 花费的时间。而前端看到的其他时间就消耗在了网络中。

在这个例子中，主要说明了响应时间怎么一步步拆。当然，如果你是下面这种情况的话，再一个个拆就比较辛苦了，需要换另一种方式。

你肯定想知道每个系统消耗了多长时间，那么我们就需要链路监控工具来拆分时间了。比如像这样来拆分：



从 User 开始，每个服务之间的调用时间，都需要看看时间消耗的监控。这就是时间拆分的一种方式。

其实不管我们用什么样的工具来监控，最终我们想得到的无非是每个环节消耗了多长时间。用日志也好，用链路监控工具也好，甚至抓包都可以。

当我们拆分到了某个环节之后，就有了下一步的动作：构建分析决策树。

构建分析决策树

关于分析决策树，我在很多场合也都有提及。

分析决策树，对性能测试分析人员实在是太重要了，是性能分析中不可或缺的一环。**它是对架构的梳理，是对系统的梳理，是对问题的梳理，是对查找证据链过程的梳理，是对分析思路的梳理。它起的是纵观全局，高屋建瓴的指导作用。**

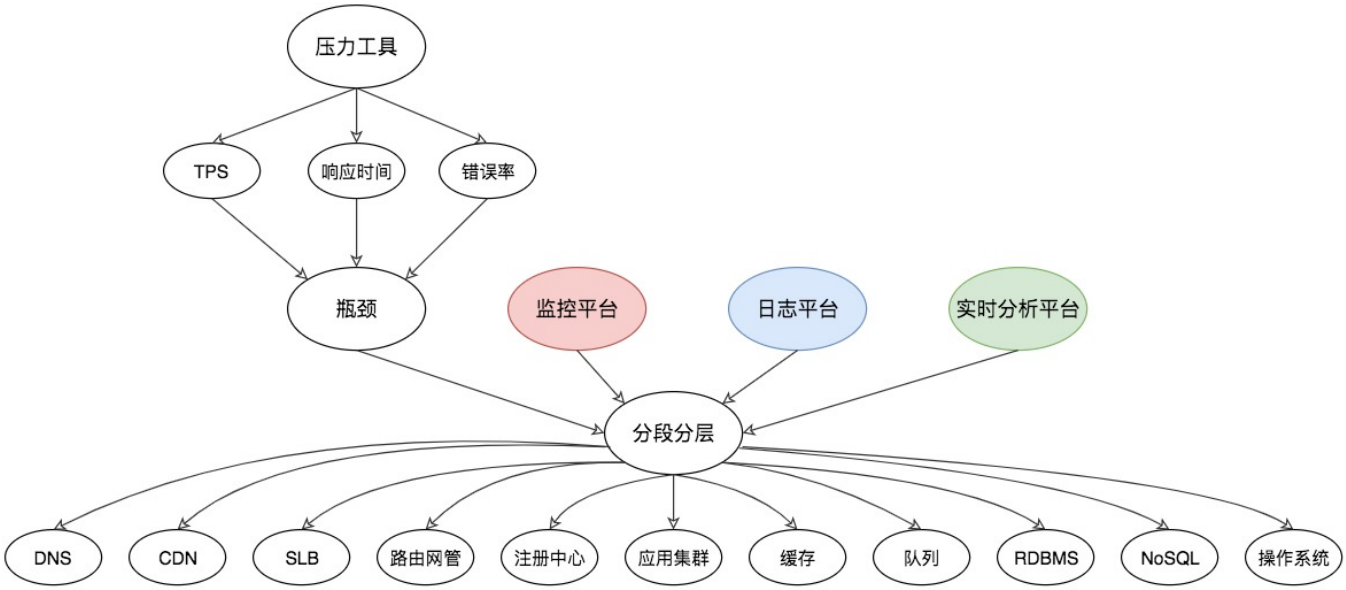
性能做到了艺术的层级之后，分析决策树就是提炼出来的，可以触类旁通的方法论。

而我要在这里跟你讲的，就是这样的方法论。

应该说，所有的技术行业在面对自己的问题时，都需要有分析决策树。再广而推之的话，所有的问题都要有分析决策树来协助。

通过上面的几个步骤，我们就会知道时间消耗在了哪个节点上。那么之后呢？又当如何？

总要找到根本的原因才可以吧，我画了如下的分析决策图：

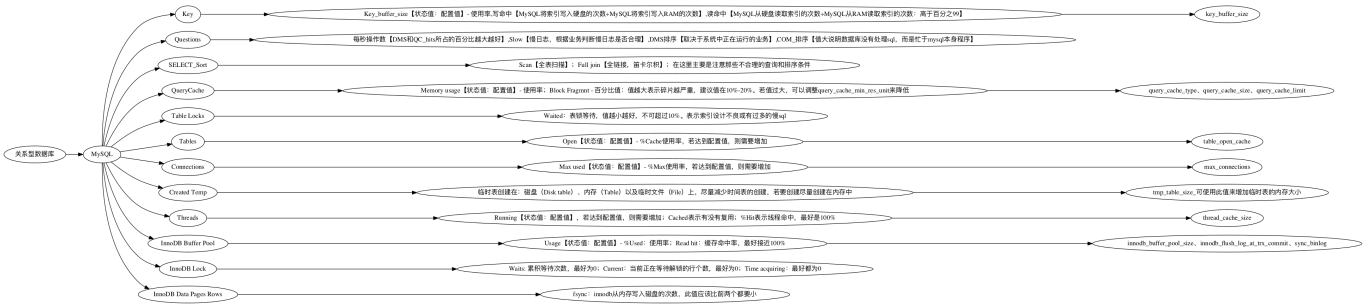


从压力工具中，只需要知道 TPS、响应时间和错误率三条曲线，就可以明确判断瓶颈是否存在。再通过分段分层策略，结合监控平台、日志平台，或者其他的实时分析平台，知道架构中的哪个环节有问题，然后再根据更细化的架构图一一拆解下去。

我在这里，以数据库分析和操作系统分析举一下例子。

首先我们看一下数据库分析决策树。

比如针对 RDBMS 中的 MySQL，我们就可以画一个如下的决策树：



由于这里面的内容实在过多，无法一次性展现在这里。我举几个具体的例子给你说明一下。

MySQL 中的索引统计信息有配置值，有状态值。我们要根据具体的结果来判断是否需要增加 key_buffer_size 值的大小。比如这种就无所谓了。

复制代码

```
1     Buffer used     3.00k of   8.00M  %Used:   0.04
```

从上面的数据可以看到，key buffer size 就用到了 4%，显然不用增加。

再比如，我们看到这样的数据：

```
1         __Tables_________________________
2      Open              2000 of 2000    %Cache: 100.00
3      Opened          15.99M     4.1/s
```

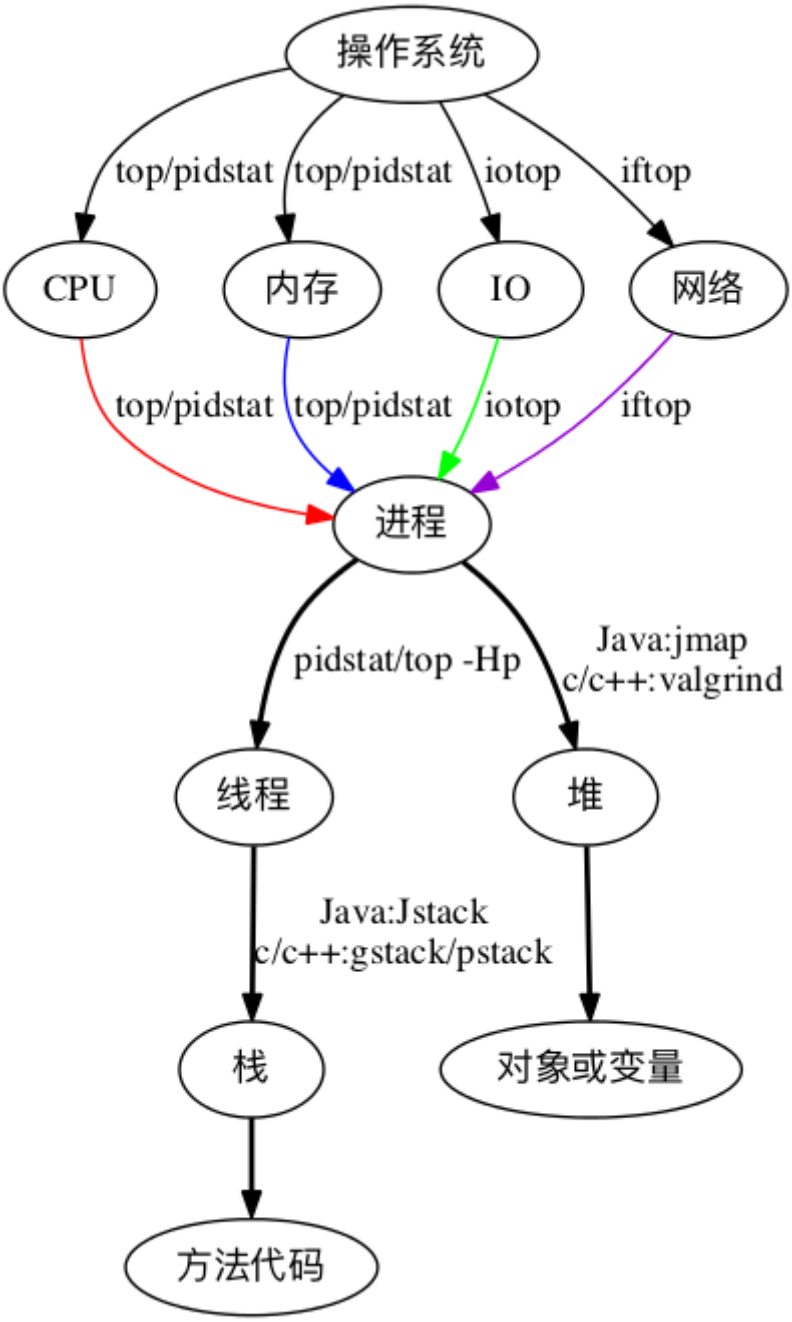
这就明显有问题了。配置值为 2000 的 Open Table Cache，已经被占满了。显然这里需要分析。但是，看到状态值达到配置值并不意味着我们需要赶紧加大配置值，而是要分析是否合理，再做相应的处理。比如说上面这个，Table 确实打开得多，但是如果我们再对应看下这一条。

```
1      Slow 2 s        6.21M     1.6/s
```

你是不是觉得应该先去处理慢 SQL 的问题了？

关于数据库的我们就不举更多的例子了。在这里只是为了告诉你，在分析决策树的创建过程中，有非常多的相互依赖关系。

然后我们再来看一下操作系统分析决策树，我在这里需要强调一下，操作系统的分析决策树，不可以绕过。



如果你想到操作系统架构图就头大，那么这时候应该觉得有了希望。那就是我觉得操作系统上的问题判断是比较清晰的，所以基于此决策树，每个人都可以做到对操作系统中性能问题的证据链查找。

但是！对嘛，总得有个但是。

对操作系统的理解是个必然的前提。我看过很多人写的操作系统性能分析方面的书籍或资料，发现大部分人把描述计数器的数值当成性能分析。

怎么理解这句话呢？比如说

“CPU 使用率在 TPS 上升的过程中，从 10% 增加到 95%，超过了预期值。” “内存使用率达到 99%，所以是瓶颈点。” “I/O 使用率达到 100%。” 等等。

像这样的描述，在我的性能团队中，一定会被骂回去重写。我要这些描述有什么用？我要的是为什么达到了这样的值，原因在哪？怎么解决？

就像分析决策树中所描述的那样，性能工程师要做的是一步步地细化分析，给出最终的原因。

有人说，如果按这个路子，似乎操作系统的分析并不复杂嘛。大概三五个命令就可以跳到代码层了。是的，对于操作来说，确实不多，但是对于判断来说，那就复杂了。举个例子来说明一下：

```
top - 14:07:51 up  3:13,  1 user,  load average: 1.40, 1.44, 1.25
Threads:   1 total,   1 running,   0 sleeping,   0 stopped,   0 zombie
%Cpu(s):  0.0 us, 40.0 sy,  0.0 ni, 20.0 id,  0.0 wa,  0.0 hi, 40.0 si,  0.0 st
KiB Mem :  8009400 total,  5344852 free,   414912 used,  2249636 buff/cache
KiB Swap:  8257532 total,  8257532 free,        0 used.  7222768 avail Mem
```

看到这样的图，你是不是有种手足无措的感觉？中断能占 40%，sy CPU 也能占 40%。这系统还用干业务的事吗？全干自己的事去了，可见操作系统有问题！你是不是要做这个判断了？

而实际情况是，这个主机上只有一个网卡队列，而请求量又比较大。

```
[root@localhost conf]# ll /sys/class/net/lo/queues/
总用量 0
drwxr-xr-x. 2 root root 0 8月  13 10:54 rx-0
drwxr-xr-x. 3 root root 0 8月  13 10:54 tx-0
```

所以要解决的是网卡队列的问题，至于怎么解决，那手段就多了。可以换个服务器，可以多加几个队列，可以多接几个节点...

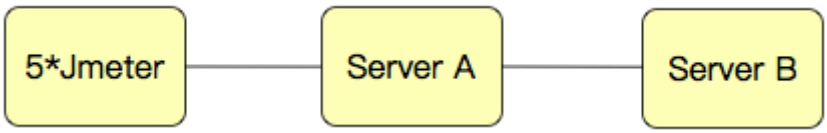
以上只是给出几个性能分析过程中常见的决策树示例。在后续的分析过程实例中，我们将秉承着这种分析思路，一步步地走到瓶颈的面前。

场景的比对

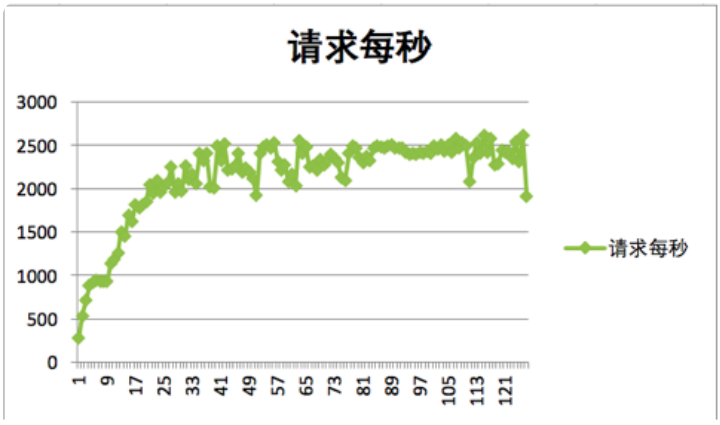
为什么要写这一部分呢？因为我看到很多人对瓶颈的判断，并不那么精确，所以想写一下场景比对的建议。

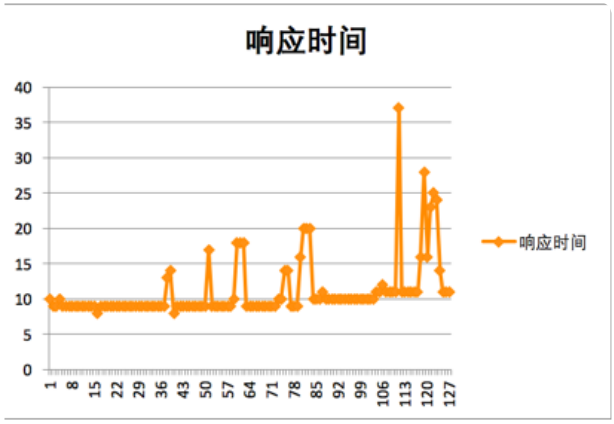
其实简单来说，就一句话：当你觉得系统中哪个环节不行的时候， 又没能力分析它，你可以直接做该环节的增加。

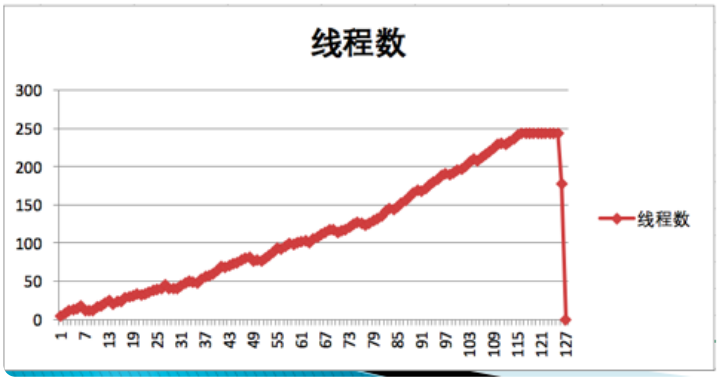
举例来，我们现在有一个如下的架构：

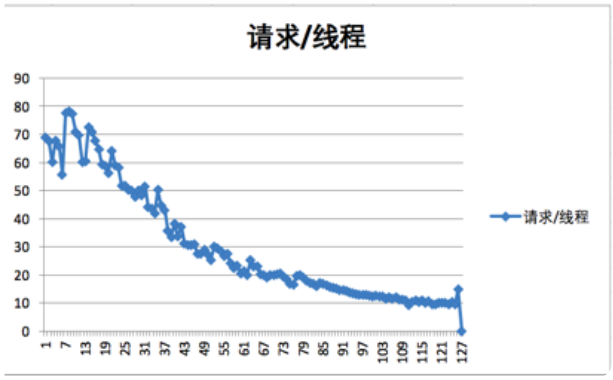


可以得到这样的结果：

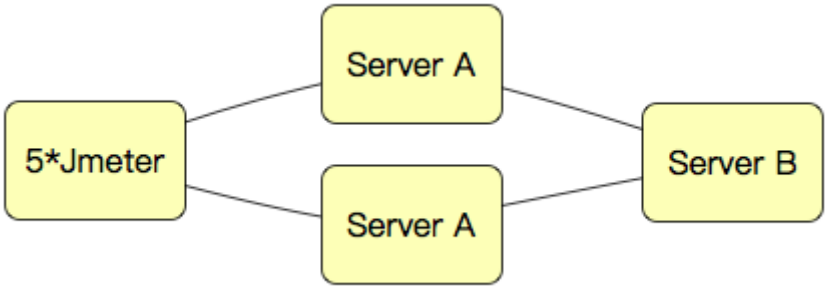




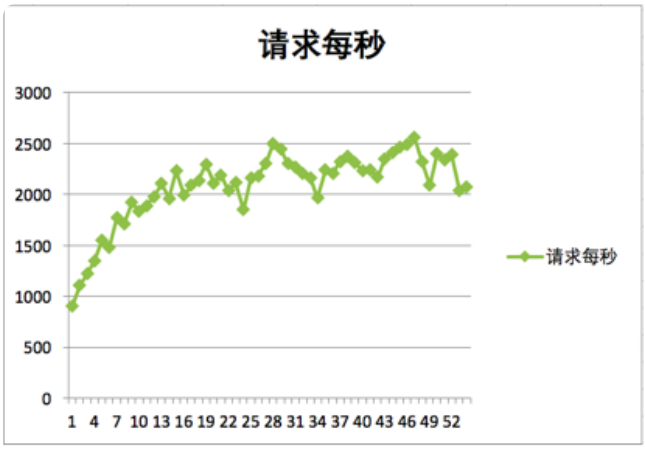


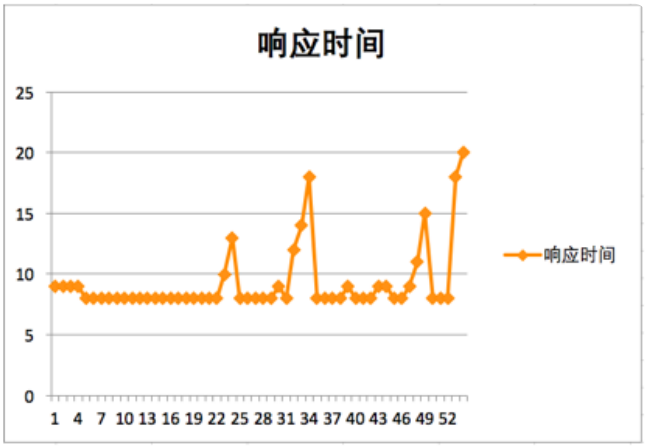


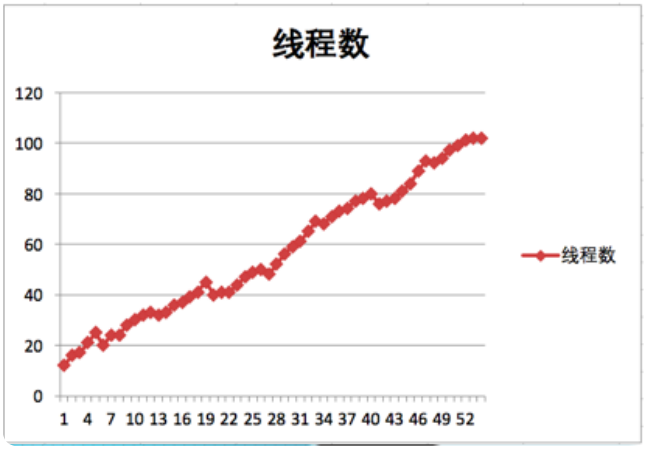
从 TPS 曲线中，我们可以明显看到系统是有瓶颈的，但是并不知道在哪里。鉴于系统架构如此简单，我们索性直接在某环节上加上一台服务器，变成这样：

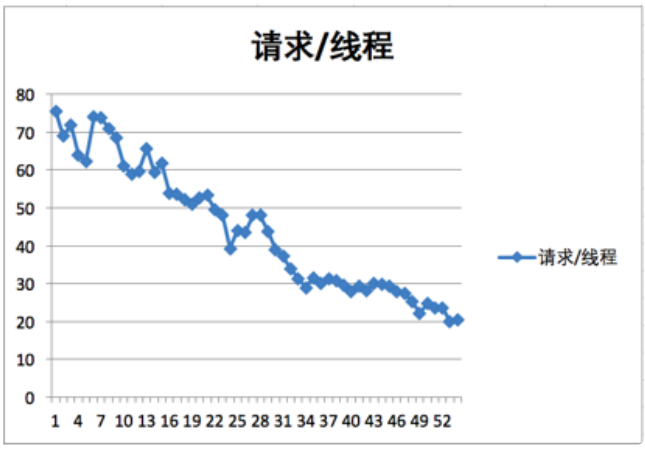


然后得到如下数据：



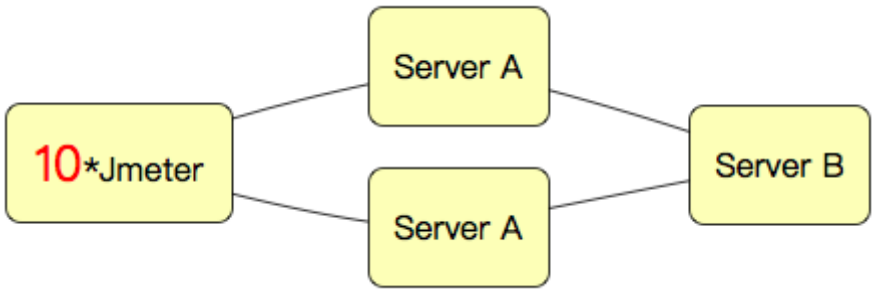




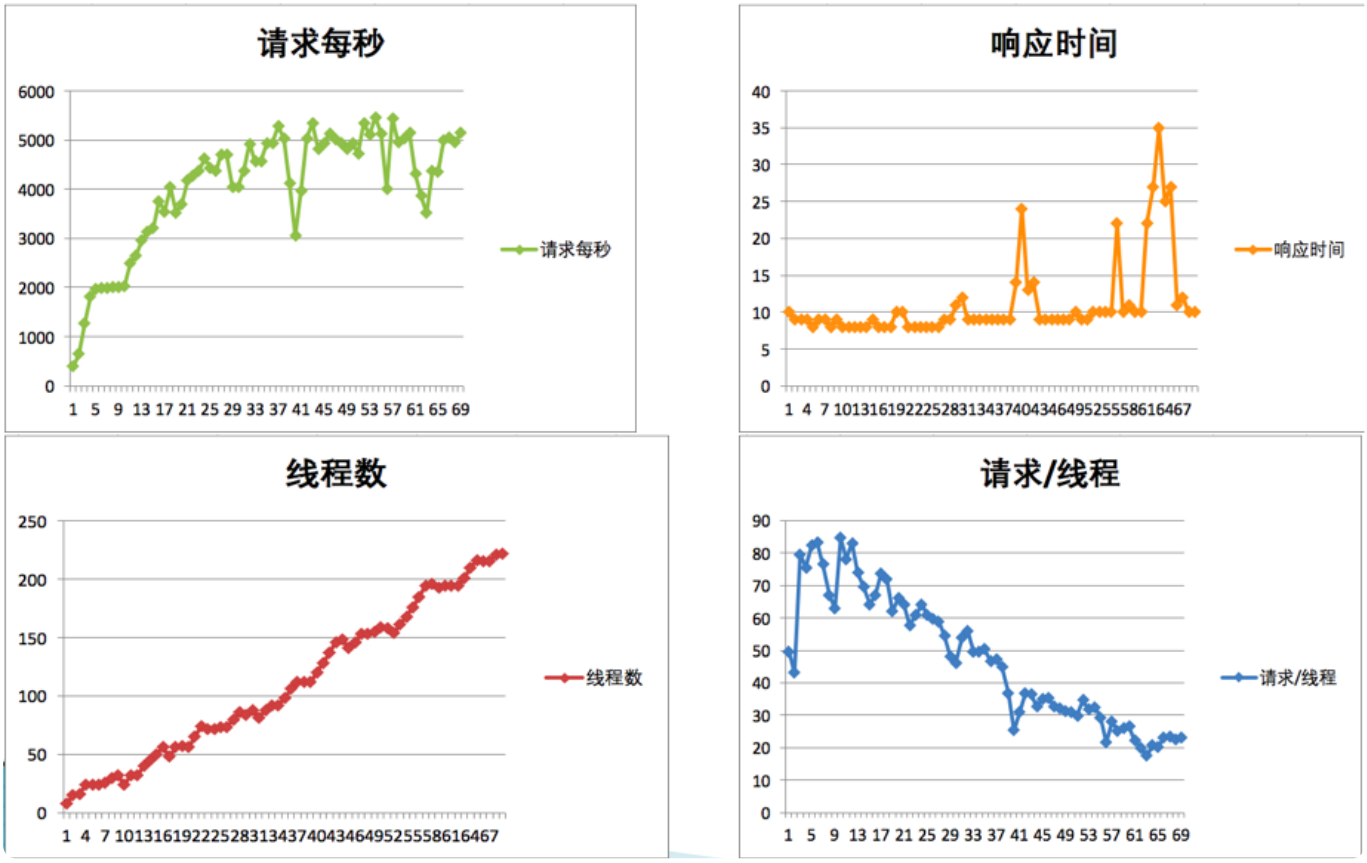


哟，没好使！

怎么办？再接着加其他节点，我加了更多的 JMeter 机器。



再来看下结果：



真巧，TPS 增加了！

看到了吧，这就是我说的场景比对。

当我们不知道系统中哪个环节存在性能瓶颈时，对架构并不复杂的系统来说，可以使用这样的手段，来做替换法，以快速定位问题。

总结

在这一篇中，我说到了瓶颈的精准判断、线程递增的策略、性能衰减的过程、响应时间的拆分、构建分析决策树以及场景的比对，这几个环节，是性能分析过程中非常重要的环节。

从我的经验上来说，这一篇文章可能是我工作十几年的精华所在了。而这里的每一个环节，又有非常多的细分，特别是构建分析决策树这一块，它需要太多的架构知识、系统知识、数据库知识等等。鉴于本文只是想起到一个提纲挈领的作用，所以无法展开描述，希望在后续的篇幅中，我们尽量细致拆解。

思考题

今天的内容虽然有点多，但总的来说，思路比较清晰，理解起来也比较容易。如果你认真学习了今天的内容，不妨思考两个问题，为什么线程递增过程不能断？构建分析决策树的关键是什么？

欢迎你在评论区写下你的思考，我会和你一起交流，也欢迎把这篇文章分享给你的朋友或者同事，一起交流一下。

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇　05 | 指标关系：你知道并发用户数应该怎么算吗？

下一篇　07 | 性能测试工具：如何录制脚本？

精选留言 (22)

写留言

@zzw
2019-12-27

第一个问题：为什么线程递增过程不能断？

这里涉及开篇提到的性能分析能力 ——「趋势分析」。
就像之前提到的一样，分析性能数据趋势需要对一个时间序列数据的分析，一般采用「线

性回归分析」算法。...

展开

3

Geek_65c0a2

2019-12-27

这节课我也期待了好久。高老师写的字数多点，总感觉不够看👀

作者回复: 编辑小美女说我文章写太长了。😣😣

3

吴小喵

2019-12-27

看到构建分析决策树就吓死了，数据库的知识，操作系统的知识都不懂啊，o(╥﹏╥)o

作者回复: 慢慢来。反正不是吓死就是累死。

2

rainbowzhouj

2019-12-27

第一个问题：不能断的原因是保证在测试过程中资源分配的合理性，减少偏差，便于分析出当前环境中的性能瓶颈点。否则断开后系统动态资源会重新分配，造成分析偏差。
第二个问题：构建分析决策树的关键好比如何画一棵树。先确定主干（主要流程），然后添枝干（组成部分），最后画树叶（定位问题）。从上到下，从左到右，拆分.....
...

展开

作者回复: 多谢肯定。
一看评论就是练家子出身的。多做总结，就会有更多的收获。

2

大拇哥

2019-12-27

做性能测试的我，对老师说的有着切身的感受，虽然此前也遇到老师所说的这些点，但从来没有像现在这样清晰，感谢~

展开

作者回复: 多谢肯定。我会再接再励。

1

qhl

2019-12-30

老师你好，在【性能衰减的过程】的示例图里面我看到，在线程达到 24 时，TPS 为 1810.6，平均响应时间是9ms。根据计算每个线程的TPS = 1000ms/9ms =111.11，111.11 * 24 个线程 = 2666，为啥显示的TPS是1810.6呢?

展开

土耳其小土豆

2019-12-30

还有另一个问题，在性能测试中，线程数是否可以衡量系统性能？比如，领导就想看线程数倒多少的时候，系统回出现错误或者崩溃，在领导看来线程数就可以衡量系统的性能，有时候我也觉得挺合理。高老师觉得呢?

作者回复: 如果你能跟领导说清楚这个线程不是真实用户，并且领导也理解了。那就无所谓。主要是不能产生误解。

土耳其小土豆

2019-12-30

再问个问题，系统的TPS为什么总是和HPS一样的呢？前端发送的请求取决于服务器的处理能力？这样的话，不就不会出现连接超时的问题了吗？ 还请楼主有时间解惑一下

作者回复: 那说明你的T定义在了一个http 请求上。如果你的T包括了多个请求的话，你再看是不是一样的值。

不过不管你怎么定义，他们的趋势都会是一样的。

土耳其小土豆

2019-12-30

看高老师的文件，感觉都能看懂，但是高老师的问题，我却回答不了，特别是第一个问题。

作者回复: 练。你还年轻。

1

村夫

2019-12-28

第一个问题，首先这样做肯定是更符合真实的业务场景，线程不断递增也有利于服务器预热。但是预热的目的是为了防止直接把服务器压死吗，就跟冬季开车前需要预热一样？这个不太清楚。而且老师说的线程递增场景的例子，当响应时间小于50毫秒，建议一个线程一个线程的往上递增。这个50毫秒我是不是要先一个线程跑个一分钟或者迭代100次来得出呢？还有就是假如真是小于50毫秒，我设计场景的时候初始线程应该是多少呢？...

展开

作者回复: 1. 在后面的场景篇会有详细的描述。先做基准场景，再做容量场景。在基准和容量中都是递增的方式。

初始化线程的多少取决于系统的承受能力，通常我会大幅度跳跃的线程数来评估系统的最大容量，然后再决定初始线程是多少。

2. 说的非常对！有这个理念就可以出师一半了。哈。

Geek_f93234

2019-12-28

第一个例子中不太明白：

针对这个结论：

“但是至少我们可以有一个非常明确的判断，那就是瓶颈在第二个压力阶梯上已经出现了。因为响应时间增加了，TPS 增加得却没有那么多，到第三个阶梯时，显然增加的 TPS 更少了，响应时间也在不断地增加，所以，性能瓶颈在加剧，越往后就越明显。” ...

展开

作者回复: 是这样的。

第一个阶梯是500tps。如果性能没有衰减。

第二个阶梯应该到1000左右，而实际的结果是没到1000，只有900左右。增幅400tps。

第三个阶梯应该到1500左右，但实际的也没到。只到了1200左右。 增幅300tps。

而在这个过程中，响应时间是不断增加的。所以性能瓶颈是明显已经出现了，并且越往后越大。

1

奔跑的栗子

2019-12-27

问题1：为什么线程递增过程不能断？
保持环境稳定干净，当性能指标出现变化时，可以排除线程突变原因
问题2构建分析决策树的关键是什么？
对架构/系统链路等足够了解，在分析问题时才能将问题逐个拆分定位；除了对架构/系统等有全面的认知外，多了解相关案例并分析熟记；

展开

作者回复: 第一问：不是保证环境稳定干净。而是为了看递增过程中会不会出现系统的动荡。
第二问理解正确。

漫漫
2019-12-27

响应时间拆分太重要了，以前都是一个系统一个系统单独调接口测试，浪费时间而且也不准确，多谢老师提供思路

展开

作者回复: 相互学习才能一直进步。
时间拆分在有些系统中简单，有些老系统日志打的差，又没有链路监控的。就费劲一点，不管怎么样，有思路总是有方法。

nana
2019-12-27

为什么线程递增过程不能断？构建分析决策树的关键是什么？
1.线程递增就是模仿线上流量逐步增加的过程，所以过程不能断
2.构建分析决策树的关键是分类，从相互独立的点去cover整个原因分析。
受益颇多，谢谢老师，期待更多干货。。。

展开

作者回复: 第二点，主要是找到正确的路径。 不跳跃，就能找到瓶颈点。

律飛
2019-12-27

思考题一，为什么线程递增过程不能断？
1.线程递增更接近真实的业务场景。性能测试的目的是为真实业务场景提供依据，提供信心，如果不能反映真实业务场景，这种测试跟没做又有什么区别呢？
2.为理解数据提供更多的信息。更多的数据信息肯定有利于更好的理解数据，防止理解偏差。...
展开

作者回复: 看到这评价，确实是认真看了内容了。理解的也很正确。

书单，我好像在其他的评论中回复了。这个比较难推荐。我的建议就是一个组件找一本优化的书完整的看完。像OS、DB、NET、代码等，各看一本，这个我没办法一一推荐。取决于各自的工作环境，工作中用什么就先看什么。其他的要看自己想学哪个。

喜哥
2019-12-27

后面图表分析 每秒请求是指每秒的请求数（RPS）吗 ，如果是，那么增加jmeter数量后，应该是每秒请求增加，而非tps增加
展开

作者回复: 用tps，还是rps，hps.....在我看来是一样的。只是叫法不同。这个在前面的篇幅中已经描述过了。

王征
2019-12-27

高楼老师这篇文章写的太棒了！
展开

作者回复: 多谢肯定。